nature portfolio

https://doi.org/10.1038/s41596-021-00593-3

Supplementary information

A metabolic modeling platform for the computation of microbial ecosystems in time and space (COMETS)

In the format provided by the authors and unedited

Supplementary Discussion 1: Detailed information on the development of the protocol

Flux Balance Analysis. Flux Balance Analysis (FBA) is a constraint-based computational method used to predict the function or phenotype of an organism by simulating its metabolism. Although it has been described extensively elsewhere[1,2], we give here a brief overview of the basic principles of FBA.

The network of metabolic chemical reactions is represented by the stoichiometric matrix *S*. In this matrix, rows represent metabolites and columns represent reactions; S_{ij} represents the moles of metabolite *i* consumed ($S_{ij}<0$) or produced ($S_{ij}>0$) by reaction *j*. FBA, like many other stoichiometry-based models of metabolism, relies on the assumption that cellular metabolism is at steady state. This assumption should be thought of as pertaining to a population of cells over a certain period of time, such that, on average, the concentrations of metabolites inside cellular biomass do not change in time. This steady state assumption imposes the following linear constraints on the fluxes through the metabolic reactions:

$$S\nu = 0$$

where v is the vector of reaction fluxes, whose *i*-th component v_i is the flux through reaction *i* (typically in units of mmol/grDW*h). Additionally, a lower (*lb*) and upper (*ub*) bounds can be set to constrain each flux between a minimal and a maximal value:

$$lb_j^\alpha \leq \nu_j^\alpha \leq ub_j^\alpha$$

These bounds may be used to define a reaction as irreversible by setting:

$$lb_j^\alpha = 0$$

In the case of "exchange reactions" (reactions representing the availability of nutrients from the environment), we use these bounds as tuning knobs to define the maximal uptake rate of the corresponding nutrients. The bulk of the metabolic fluxes are left virtually unbounded. Thus, in practice, the main constraint to internal metabolic fluxes arises from the requirement of mass balance, defined by the stoichiometric matrix and ultimately by the structure of the metabolic network, and by the boundary conditions of nutrient availability and thermodynamic infeasibility. Note that, as described later, the nutrient availability flux bounds will be dealt with in a substantially different way in dynamic FBA and in COMETS.

The maximal uptake rate for the exchange reactions is typically modeled with the Michaelis-Menten kinetics:

$$lb = V_{max}\frac{[C]}{[C]+K_M}$$

where $[C]$ is the nutrient concentration, V_{max}Vmaxis the maximum rate and K_M KM is the Michaelis constant.

Mathematically, these constraints define a convex polytope, i.e. a volume of permitted fluxes in high dimensional space, with the number of dimensions defined by the number of reactions, i.e. the number of columns of the stoichiometric matrix *S*. Note that the reconstruction of a metabolic network from an organism's genome (described in detail elsewhere[3]) involves substantially more complicated steps, including a detailed mapping between genes and reactions. These steps are not described here, but are an important component for the usage of FBA methods towards making accurate predictions.

To predict a specific set of fluxes for a given metabolic network, FBA requires an additional step, in which the feasible space is searched for a point (or set of points) that maximizes (or minimizes) a given objective function, represented in the form of a linear combination of the flux variables. Usually, this objective function is the production of a set of molecules (building blocks, energy and redox currency) that metabolism needs to provide in precise proportions as required by other cellular processes (synthesis of macromolecules, membranes, DNA replication, transcription, etc.) to generate new biomass[4]. The use of linear objective functions makes it possible to solve this mathematical problem through well-established efficient linear programming algorithms, available through a number of libraries. A typical FBA optimization for a genome-scale model, on a standard laptop computer, takes on the order of a few milliseconds. Biologically, the search for a set of fluxes that optimizes a given objective implies the hypothesis that an organism has evolved to be able to regulate its metabolic fluxes to approach that optimum under a set of environmental conditions. In other words, the model assumes an "optimal regulation". This assumption is partly justified by evolution[5,6], but it does not necessarily hold in all conditions[5,7–9]. COMETS can accommodate arbitrary objective functions, in addition to maximization of biomass production. Moreover, it supports multiple objectives optimized [5,10] iteratively[5,10], including the minimization of the sum of the absolute values of fluxes (also known as parsimonious FBA)[9].

Dynamic Flux Balance Analysis (dFBA). Dynamic Flux Balance Analysis[11] is an iterative extension of FBA that explicitly includes the dynamics of the organisms as they grow, and the effects of this growth in the environment. dFBA produces piecewise-linear approximations of the microbial growth curve (i.e., biomass as a function of time), and of the environmental abundance of metabolites, that can change due to external factors, or through uptake/secretion fluxes. Notably, in dFBA, while extracellular metabolites can dynamically change, intracellular ones are still assumed to be at steady state (through fast equilibration). In COMETS, for each microbial species α, we implement dFBA by numerically solving its biomass equation:

$$\frac{\partial B^\alpha}{\partial t} = \nu^\alpha_{growth} B^\alpha$$

where B^α is the biomass of species α and v^α_{growth} is the growth rate, as computed through FBA. Effectively, upon fixing a finite Δt, a change of biomass for each species in the next time step is computed as $\Delta B^\alpha = v^\alpha_{growth} B^\alpha \Delta t$. The dynamics of each external metabolite is governed by the equation:

$$\frac{\partial Q^i}{\partial t} = \sum_\alpha \nu^\alpha_i B^\alpha$$

where Q^i is the abundance of external metabolite i and v^α_i is the exchange flux of metabolite i in species α. Similar to the biomass equation, the changes in extracellular metabolites are computed based on FBA-inferred fluxes and the finite time interval.

At the beginning of the simulation, the starting molecular composition of the environment is initialized, based on initial conditions set by the user. At each iteration, the program estimates each model's uptake bounds based on the external concentration of nutrients, and solves each model's FBA, obtaining an estimate of the growth rate and all other fluxes, including uptake and secretion. Because models are optimized sequentially, when a nutrient is limiting, if there are many models present in the simulated layout, the total amount of a given nutrient scheduled to be uptaken in a single time iterationof a given nutrient will likely may exceed the actual amount present in the system. This is due to the fact that the calculation of the nutrient uptake rate in each model separately takes into account its total concentration. If several models are present, several uptake rates will be calculated for the same nutrient.its concentration in presence of multiple populations (models). To prevent this artifact,

COMETS checks at every iteration whether the total uptake, the sum of uptakes for a given nutrient from all models, is higher than the total amount presentconcentration for anythe said nutrients. If this happens, if there is no sufficient amount of a nutrient to be uptaken according to the total uptake rate, COMETS performs a second optimization, this time adjusting the uptake of all models to be proportional to the uptakes computed in the first optimization without exceeding nutrient concentration in the environment. The resulting fluxes are used as inputs in the above equations to compute the changes in biomass and extracellular metabolites. One of the important outcomes of this process is the fact that different organisms may compete for common resources and/or exchange metabolites as an outcome of their own objective function. Microbe-environment and microbe-microbe interactions are emergent properties of the physiology of each species[12].

As mentioned above, a key aspect of dFBA is that it requires a mapping between the external nutrient concentration and the maximal uptake rate for each metabolite in each organism. COMETS includes three possible choices for such mapping functions. Bounds can be either a linear function of the concentration, a Monod (or Michaelis-Menten) function, or a pseudo-Monod uptake type (i.e., linear until a given threshold, then constant). The type of uptake can be specified in the parameter exchangeStyle. A special case of "nutrient" uptake is light absorption, which is calculated from absorption coefficients using the Beer-Lambert law.

Metabolite control. All simulations begin with an initial metabolite environment, which may vary across space. Metabolites can also be set to change in pre-defined ways during the simulation. A metabolite can be assigned the static property, which causes it to begin each time step at the defined value. Second, the refresh property can be used to add (or remove) a constant amount of metabolite to a spatial location per-hour, divided equally among the time steps. Third, metabolite abundances can be set to vary periodically using defined wave functions. Finally, all metabolites can be set to dilute proportionally, using the parameter setting metaboliteDilutionRate.

Spatial structure and dynamics. The classical implementation of dFBA described above (which can be implemented in COMETS) corresponds to a well-mixed system, in which all microbes and metabolites are uniformly distributed and have access to each other in proportion to their concentration. In addition to this dynamics in time, COMETS is able to take into account the spatial structure of microbial colonies and communities, simulating arbitrary two-dimensional spatial structures (a 3D version is in principle available, but has not been thoroughly tested yet). Spatial structure in COMETS is implemented as a 2D grid of cubic "boxes" with a given dimension and volume. Inside each of these "boxes", a well-mixed scenario is assumed. The biomass of different species and the environmental metabolites can propagate from a given box to neighboring boxes based on physics laws of convection-diffusion, as described in detail below.

Biomass propagation. The core of the COMETS method is the simulation of the propagation of the biomass present in the system. The simulations are performed by numerically solving the partial-differential equations that govern the dynamics of the system [13–18]. The dynamical variable of biomass (formally biomass density) is spatially continuous. Although the natural unit of biomass is a single cell of an organism, we implemented the biomass dynamics as one of a locally averaged continuous quantity. The reason for this choice is to be able to simulate macroscopic systems on the order of centimeters and larger. An individual cell-based methodology [19,20] would significantly hinder the extent ofin both size and time of the simulations.

The partial differential equation for biomass propagation written in the general form is:

$$\frac{\partial B^\alpha}{\partial t} = \vec{\nabla}(D^\alpha \vec{\nabla} B^\alpha) - \vec{\nabla}(B^\alpha \vec{u}^\alpha) + f^\alpha(B^\alpha, Q^m)$$

Here $B^\alpha = B^\alpha(\vec{r},t)$ is the biomass of species α at spatial position $\vec{r}$ and at time t. The operator $\vec{\nabla}$ is the vector differential operator, $D^\alpha = D^\alpha(\vec{r},t)$ is the diffusivity of species α, and it can vary in space and time explicitly, or as a function of the local biomass. $Q^m = Q^m(\vec{r},t)$ is the local nutrient/metabolite content (density). $\vec{u}^\alpha = \vec{u}^\alpha(B^\alpha;\vec{r},t)$ is the local velocity of the bulk biomass of the corresponding species. The biomass velocity can be a function of the biomass (as a mechanistic model) or explicitly a function of the time and spatial position. Finally, $f^\alpha(B^\alpha,Q^m)$ is the biomass growth/death term. This term has the same form as the corresponding one for dFBA.

The temporal dynamics of the biomass at a spatial point is governed by the three terms on the right-hand side of the equation. The first term is a diffusive one, and it models the free movement of the individual bacterial cells. The diffusivity may be an explicit function of time and/or spatial position. In this case the local diffusivity depends on the external conditions, such as material in the region where the biomass is propagating, etc. The diffusivity may also be a function of the biomass, modeling the cooperativity in the propagation of the bacterial colony. The second term on the right-hand side of the equation is the advective one and models the bulk motion of the biomass with a local velocity $\vec{u}^\alpha$ u. The local velocity may explicitly depend on the spatial point and time. This would be a model of biomass motion in external flow. The biomass velocity however may be a function of the biomass itself given via a mechanistic model, such as a model of propagation by mutual mechanical pushing of the cells. By combining these terms for biomass propagation, we can model a wide range of modes of bacterial motility[21]. Setting the diffusivity to a constant and the convective term to zero, we can model, for example, simple diffusive swimming and twitching[21]. Having the diffusivity be a function of the biomass itself, we model the collective motion of the bacteria. The advective term models the sliding of the colony due to mechanical pushing of the cells during the colony expansion[21].

In COMETS we have implemented the mechanistic model of biomass propagation by cellular pushing [14]. As individual cells grow and divide, the local density of the biomass is increased. At the point when the density reaches the value of densely packing, the cells are in mechanical contact, and a field of pressure develops due to the mechanical interaction, i.e. pushing of neighboring cells. The local velocity of biomass B^α is given by the gradient of the local pressure developed due to cells pushing on each other:

$$\vec{u}^\alpha = \vec{\nabla} P / \mu^\alpha$$

where μ^α is a friction constant and P is the pressure field given by:

$$P = E^\alpha (1 - \rho_0/\rho)^{3/2}$$

where ρ is the local density of the total biomass, E^α is the elastic constant for species α and ρ_0 is the density of the biomass at closed packing, i.e. when the bacterial cells are touching, but not pushing each other. If the density $\rho<\rho_0$, the pressure field is equal to zero.

Another model for propagation of bacterial biomass that we implemented in COMETS simulates the cooperative behavior in a dense bacterial colony, based on the fact that certain bacterial species secrete a lubricant[17,22] that changes the local mobility of the bacterial cells. This secretion is typically dependent on the local density of cells. In COMETS we simulate this phenomenon by modeling the biomass diffusivity of species α as:

$$D^\alpha = D_0^\alpha + D_k^\alpha \rho^k$$

where D^α_0 is a general linear term, and $D^\alpha_k \rho^k$ depends on the local biomass density to the power of k.

In addition to the dependence of D^{α} on the biomass density, the diffusivity of the biomass may optionally be restricted to the parts of the biomass field that are actively growing. We implement this feature by multiplying D^{α} with the Hill function:

$$H(\Delta B^{\alpha}) = (\Delta B^{\alpha})^{n}/((\Delta B^{\alpha})^{n} + K^{n})$$

where ΔB^{α} is the local biomass change due to growth in a single discrete time step, and *n* and *K* are the Hill function parameters.

The parameters of the models for the biomass propagation can be set separately for each model in a COMETS simulation. The biomass propagation parameters are independent of the metabolic properties of a model, and can be set separately in the model input file. This way, one can include in a simulation either (i) species that are metabolically different, but have similar or identical biomass propagation properties, or (ii) species that are metabolically identical, but differ in their physical properties, or (iii) different strains with completely identical metabolic and physical properties, as we illustrate in Procedure 7.

The last term in the equation for biomass propagation models the local growth (and/or death) of the biomass. Typically, this term is proportional to the biomass, and a pre-factor given by the growth rate of the biomass. In COMETS, the growth rate of the local biomass is determined by the metabolic activity in the model, and is a function of the local quantity (formally concentration) of the external nutrients/metabolites. In COMETS, we calculate the growth rate utilizing the Flux Balance Analysis methodology (as described above). The local growth rate can also be augmented by a death rate that removes a fraction of the biomass at each time step.

Nutrient propagation. In addition to the biomass propagation, COMETS simulates the spatio-temporal dynamics of the metabolites/nutrients that are taken up and/or secreted by the organisms. The dynamics of the external metabolite m is determined by their uptake and secretion by the organisms, as well as their convection over the spatial layout:

$$\frac{\partial Q^{m}}{\partial t} = \vec{\nabla}(D^{m}\vec{\nabla}Q^{m}) - \vec{\nabla}(Q^{m}\vec{u}^{m}) + \sum_{\alpha} B^{\alpha}\nu^{\alpha,m}$$

Here the diffusivity of the metabolites may also be locally defined, or/and can depend on the local biomass content.

Barriers. Propagation of biomass or nutrients can be prevented into certain lattice locations by the placement of barriers. Barriers act as reflective boundaries for diffusion and biomass motion calculations. In the python toolbox, a helper function ("grow_rocks", see Procedure 6) is included to create a common clustered barrier, which works as follows: grow_rocks first picks n random locations within the allowed range. Then, mean_size * n - n additional points are added to these seed locations. For each new point, one random location out of the set of all possible unoccupied locations next to occupied locations are chosen. If an unoccupied location is adjacent to >1 occupied location, its chance of being chosen is increased by 100% for each occupied adjacent location. Only lattice points directly to the NSEW of occupied locations are considered. This process is repeated until all new points are assigned.

Demographic and growth noise. Two types of stochastic noise are implemented in COMETS. Growth rate noise consists of a simple broadening of the growth rate with a Gaussian noise term. Instead of implementing the growth rate as calculated by the FBA algorithm, it is sampled from a Gaussian distribution centered at the FBA obtained growth rate. The user should be aware that this broadening is

meant to be very small, since it may lead to a growth rate that is temporarily higher than the FBA maximum. While this noise will average out in a sufficiently long simulation run the user may want to verify that the variation due to this noise is indeed small in each simulation step. When applying this type of noise the user should also be aware that the uptake of nutrient is proportional to the biomass, and it will vary proportionally to the applied noise.

Demographic (shot) noise is given as a stochastic term in:

$$\frac{\partial B^\nu}{\partial t} = ... + \mu B^\nu + \sigma\sqrt{B^\nu}\eta$$

where B^ν is the biomass of species ν, η is white noise and σ is a parameter that determines the magnitude of the noise. The demographic noise is implemented in COMETS according to the method described in [23].

The change of the biomass from the growth term is first calculated, then it is resampled in two steps according to the procedure in[23]. First, we sample the shape parameter of the Gamma distribution from the Poisson distribution:

$$P(\alpha;\lambda) = e^{-\lambda}\frac{\lambda^\alpha}{\alpha!}$$

where:

$$\lambda = \frac{2B^\nu}{\Delta t(\mu\sigma)^2}.$$

Then, with the sampled α, we sample from the Gamma distribution:

$$\Gamma(x;\alpha,\beta) = \beta^\alpha\frac{x^{\alpha-1}e^{-\beta x}}{\Gamma(\alpha)}$$

where the scale parameter $\beta=1$ and $\Gamma(\alpha)$ is the Gamma function.
The new biomass is given by:

$$B^\nu = \frac{1}{2}\Delta t(\mu\sigma)^2 x.$$

Extracellular reactions. COMETS includes the capability to use kinetic rate laws to simulate two types of reactions involving extracellular media components. The first are elementary reactions of arbitrary order with any number of reactants or products, of the form

$$x_1M_1 + x_2M_2 + \rightarrow y_1P_1 + y_2P_2 + ...$$

for reactants M and products P, with respective stoichiometries x and y, and with a reaction rate

$$r = k\Pi[M_i]^{x_i}$$

given the rate constant k. The second type are enzyme-catalyzed reactions with a single substrate and any number of products, of the form

$$S \rightarrow y_1P_1 + y_2P_2 + ...$$

where the stoichiometry of the substrate S is always assumed to be 1. The reaction rate is determined by the Michaelis-Menten equation

$$r = k_{cat}[E] * [S]/(K_M + [S])$$

that accounts for the concentrations of the enzyme E and the substrate S the turnover rate k_{cat}, and the half-saturation concentration K_M.

Changes in metabolite concentrations over the course of a single simulation timestep are calculated by converting the set of all extracellular reactions into a system of ordinary differential equations, then approximating the solution with the classical Runge-Kutta integrator from the Apache Commons Math library (http://commons.apache.org/). The process of updating metabolite concentrations by applying the effects of extracellular reactions happens once during each simulation timestep, after metabolites have been updated by the dFBA process and before diffusion occurs.

Random mutation. In addition to ecological dynamics, COMETS also has the capability of mutating species during the simulation, which results in the capability of simulating evolutionary dynamics. Mutations occur during growth: at each iteration and for each species α, COMETS first computes the number of new individual cells arising in the previous time interval Δt as

$$N_G = [B_\alpha(t) - B_\alpha(t - \Delta t)]\, C_S$$

where C_s is the size of a single cell (in grams of dry weight, specified by the parameter cellSize). Given the total population growth N_g and mutation rate μ, COMETS stochastically samples a number from a Poisson distribution with mean $N_g\mu$ (or a binomial if populations contain less than 10 cell divisions). The resulting mutants - new stoichiometric models with modified stoichiometry based on a set of rules (see below) - are then placed randomly in cells containing biomass of the ancestor, with a probability per cell of the simulation grid proportional to the fraction of N_g in that cell. The new mutant populations are also mutable with the same mutation rates as the ancestor, allowing the accumulation of mutations in time.

Two types of mutations are implemented in COMETS, reaction knock-out and reaction knock-in. The knock-out rate μ_{ko} is set using the parameter mutRate, and represents the knock-out rate per generation and per reaction. Thus, $\mu = R\mu_{ko}$, where R is the number of reactions of a given model. In contrast to knock-outs, the knock-in rate is computed per generation and is set up using the addRate parameter. In order to simulate knock-in mutations (i.e. reaction additions) models must be previously prepared by adding all the reactions that we want to be potential additions to the model or models, with both upper and lower bounds equal to 0. These reactions will initially be unavailable to the optimizer, and become available only once "added", i.e. once their upper bound is set to 1000 by COMETS during the simulation. Future plans include implementing mutations in genes, that would propagate to reactions using gene-to-reaction logical relationships.

Numerical integration of spatio-temporal equations. The method used for numerical integration of the partial differential equations in COMETS depends on the type of equation, i.e. the type of model of spatio-temporal propagation, that is being solved. The three different models for propagation of biomass, the simple diffusion, propagation by pushing and non-linear cooperative diffusion, cannot be optimally solved by a single method.

For the simple (linear) diffusion model of biomass propagation the user can choose between two implemented numerical methods for its solution. One is using an alternating direction implicit (ADI) scheme with a central difference formulation[12] and the other is an 8-point integration scheme. The other two models of biomass propagation, the model of convection (pushing) and the non-linear diffusion, due to the presence of the nonlinear terms, are solved by implementing the predictor-corrector Adams-Bashford-Moulton scheme[24,25]. The diffusion of the media is solved by the standard implicit method, the same as for the linear diffusion of the biomass.

Supplementary Discussion 2: Software architecture and the basics of using COMETS

Software architecture of COMETS

Core architecture

The core of COMETS is written in Java and performs its main functionalities: dynamic FBA, propagation of biomass and media in time and space, extracellular reactions or evolution.

The code is organized in several Java packages, each containing several classes. The core of the code is in the following packages:

comets: This package contains the top-most super-classes and interfaces. This organization of the superclasses was done with a future development in mind, including the possibility of modules that compute growth with algorithms other than FBA.

fba: This package contains the core of the program, including most of the data structure as well as the run methods that perform the core procedures of the simulation. Here are most of the FBA specific subclasses of the superclasses found in the comets package.

ui: This package contains the classes related to the graphical user interface.

util: The util package contains general mathematical utilities, independent of the FBA methodology, such as several PDE solvers.

The Java core of the FBA methodology in COMETS is structured in four main classes, organized hierarchically:

Model: Instances of this class are metabolic models which are optimized using FBA in the simulations. This class also has a method to mutate models, i.e. add or remove reactions, used for evolutionary simulations.

Cell: The spatial structure in COMETS is structured as a grid (either 2 or 3-dimensional). The cell class represents what we refer to earlier as a "box", i.e. a single location on this grid, with defined dimensions. Note that this cell should not be confused with a biological cell. A cell class contains models and media, whose biomasses and concentration are updated in each iteration of the dynamic FBA simulation. This class also performs the extracellular reactions, if present.

World: The world contains all the cells (i.e. boxes, see cell class above) with their models and media. After each iteration, this class performs the computations necessary to propagate biomass and media

between cells. In evolutionary simulations, it also decides stochastically which models mutate and in which cells.

Parameters: This class contains all the necessary parameters for running a simulation. These parameters, their units, and default (or alternative) values are listed in the Table in Appendix 2.

Comets: This is the main class of COMETS. It integrates all of the above, runs a simulation and produces the output.

The software contains some additional "helper" classes that deal with file loading or optimizers used by the simulation. The central class that does the FBA optimization is the abstract class FBAOptimizer, which has two subclasses, FBAOptimizerGurobi and FBAOptimizerGLPK.

The basics of COMETS using the MATLAB toolbox

The COMETS MATLAB Toolbox is a collection of classes and functions intended to facilitate the processes involved in creating layouts for simulations, and includes utilities to execute COMETS within scripts from the command line and to parse output files. Similarly to the Python Toolbox's use of COBRAPy, the MATLAB Toolbox uses metabolic models in the format of the COBRA Toolbox for MATLAB[3].

A brief overview of the most important components of the COMETS MATLAB Toolbox follows. For more in-depth documentation, see http://segrelab.github.io/comets-toolbox/ and https://segrelab.github.io/comets-manualhttps://comets-manual.readthedocs.io.

Primary Tasks

Manipulating metabolic models: The MATLAB Toolbox uses stoichiometric metabolic models in the format of the popular COBRA Toolbox for MATLAB[3], allowing users familiar with COBRA to quickly get up to speed, and allowing us to begin from model source files in an already wide-spread format. Because COBRA is not intended to support dynamic Flux Balance Analysis, we have added fields to the model structure that capture temporal behaviors: the setBiomassRxn() function can be used to specify a reaction that determines the growth rate, and the setKm() and setVmax() functions allow individual uptake reactions to be given a rate that is dependent on metabolite concentrations in a Michaelis-Menten-like manner.

Testing if COMETS is functional: The command testComets() will check, if COMETS related environmental variables were set properly, and if COMETS can be called from Matlab.

Creating layouts: A "layout" is the structure which represents the simulated world in a COMETS analysis, containing sets of metabolic models with individual biomasses, as well as metabolites, across the space of a 2-dimensional grid. A layout object contains a CometsParams object which can be saved and loaded

in order to conveniently ensure that all simulations are performed with a user's preferred set of simulation parameters and default values. The layout also contains instructions for the addition or removal of media over the course of the simulation, the diffusion properties for each organism and metabolite, the kinetic parameters of extracellular reactions (such as decay or enzymatic degradation), and the locations of "barrier" spaces which block diffusion. Utility functions exist to automate the placement of some layout elements, for example placing bacterial colonies at equidistant points or applying barrier spaces at the edges of the grid in order to create a circular plate.

Creating COMETS inputs and executing simulations: The formats of the text files required by COMETS for models and layouts are not amenable to editing by hand, as doing so requires the user to track the indexes of multiple elements in several lists and refer to the documentation for the details of each field. The COMETS MATLAB Toolbox provides scripts to generate these files so that users no longer have to concern themselves with the contents of these files, and can save input files sets using the createCometsFiles() command or directly execute a simulation using a layout object with the runComets() command.

Handling COMETS outputs: The various log files generated by COMETS for biomass, metabolite concentrations, and fluxes can be loaded into MATLAB tables for easier filtering and analysis through the functions parseBiomassLog(), parseMediaLog(), and parseFluxLog(). Utility scripts are provided which make it simple to generate plots of these measurements over time with the functions plotBiomassTimecourse() and plotMediaTimecourse().

Classes and Data Structures

CometsLayout: The main class which encapsulates all information involved in a single COMETS simulation by containing media contents, the list of COBRA models, spatial information, and a single CometsParams object. Contents of the layout should be manipulated by using the methods of the CometsLayout class instead of being directly modified when possible, for example by editing initial media through setMedia() or adding metabolic models through addModel().

CometsParams: A class to contain the parameters for global simulation parameters as well as model-level default values.

File I/O

createCometsFiles(cometsLayout, [directory, layoutFileName, separatePamsFiles]): Creates the COMETS script, layout, and model files. If separateParmasFiles is true, it creates separate files to contain the global and package parameters. Otherwise, all parameters are included in the body of the layout file.

parseBiomassLog(fileName): Processes a MATLAB-format biomass log from a COMETS simulation. Returns a table with the following columns:

- t: Timestep.

- X: X coordinate.
- Y: Y coordinate.
- Z: Z coordinate. Excluded in 2D simulations.
- model: ID number of the model. Arranged as in the layout file, starting with 0.
- biomass: biomass value in grams.

parseFluxLog(fileName): Processes a MATLAB-format reaction flux log from a COMETS simulation. Returns a table with the following columns:

- t: Timestep.
- X: X coordinate.
- Y: Y coordinate.
- Z: Z coordinate. Excluded in 2D simulations.
- model: ID number of the model. Arranged as in the layout file, starting with 0.
- rxn: ID number of the reaction. Arranged as in the metabolic model, starting with 0.
- flux: Flux through the reaction.

parseMediaLog(fileName, [metNames]): Processes a MATLAB-format media log from a COMETS simulation. If a cell array is provided as metNames, only records for the corresponding metabolites will be loaded. Returns a table with the following columns:

- t: Timestep.
- X: X coordinate.
- Y: Y coordinate.
- Z: Z coordinate. Excluded in 2D simulations.
- met: ID number of the metabolite. Arranged as in the layout file, starting with 0.
- amt: Concentration of the metabolite in millimoles.
- metname: Name of the metabolite.

Standard Workflow

createLayout({cobraModels}): Initialize a COMETS layout with default properties as stored in the CometsParams class, and add any metabolic models provided as arguments as though invoking CometsLayout.addModel().

addModel(*cometsLayout, cobraModel*): Attaches the given model to the layout, and adds any of the model's exchange metabolites to the layout's list of media components.

setMedia(*cometsLayout,{metaboliteNames},concentrations*): Set the initial media concentrations for every space in the simulation. To alter the initial media in an individual grid cell, use the method setInitialMediaInCell() instead.

setInitialPop*(cometsLayout, [format, amount, resize])*: Sets the initial population for each metabolic model in the given *cometsLayout*, arranged according to the *format* parameter, to the value or values

provided in the amount parameter in grams. If *resize* is not false, the dimensions of the *cometsLayout* will be adjusted as well. *Format* may be one of two options:

- ‘Colonies’ (default): Up to four evenly spaced colonies will be created, one for each model in the cometsLayout. Default dimensions 100 by 100 grid cells.
- ‘1x1’: Biomass for all models will be placed in the center grid cell. Default dimensions 1 by 1 grid cell.

RunComets(cometsLayout, [directory]): Creates layout and model files in the given directory (defaulting to the current working directory) and executes a simulation by invoking COMETS through the command line.

The basics of COMETS using the Python Toolbox

We will first walk through the basic functionalities of COMETS using the Python Toolbox, and more specific examples of usage will be provided in the next sections. Once cometspy has been installed, a user can implement this and all other python protocols in one of two ways: 1) The user can copy the code into a python script and run it. 2) We have also provided the protocols in jupyter notebook format. To use these included files, start the jupyter notebook by typing the following command line:

```
jupyter notebook
```

In Windows this is best done by going to the start menu, and running "Anaconda Powershell Prompt". The above command can be run from the Anaconda Powershell.

This will launch a browser tab. On this tab, load the corresponding protocol through file-open and finding the [name of protocol].ipynb file (for example, chemostat.ipynb). To run each jupyter notebook click on the kernel tab and then click Restart & Run All. Warning: for some protocols, this may take from five minutes to several hours.

Create of the COMETS input files

1. Import the python comets toolbox, the cobra toolbox and the cobra.test tools.

```
import cobra
import cobra.test
import cometspy as c
```

2. Load an existing model using COBRAPy. Here, we use the custom function cobra.test.create_test_model() from the COBRAPy toolbox to load the *E. coli* model.

```
# Load a textbook example model using the COBRAPy toolbox
test_model = cobra.test.create_test_model('textbook')
```

3. Use the loaded COBRA model to build a COMETS model class, which allows us to change COMETS-specific model parameters, such as initial population sizes.

```
# Use the above model to create a COMETS model and open exchanges
test_model = c.model(test_model)

test_model.open_exchanges()

# Change comets specific parameters, e.g. the initial biomass of the model
test_model.initial_pop = [0, 0, 1e-7]
```

4. Use the params class to generate a list (i.e. a python dict object) containing the default parameter values.

```
# Create a parameters object with default values
my_params = c.params()
```

5. Change the parameter values as desired

```
# Change the parameter "maxCycles" corresponding to the number of iterations in our simulation
my_params.set_param('maxCycles', 100)
```

6. Check which other parameters are available and their current value.

```
# See available parameters and their values
my_params.show_params()
```

7. Use the layout class to generate a layout with the previously prepared model as input. Then, add minimal media components.

```
my_layout = c.layout(test_model)
my_layout.set_specific_metabolite('glc__D_e', 0.011)
my_layout.set_specific_metabolite('o2_e',1000);
my_layout.set_specific_metabolite('nh4_e',1000);
my_layout.set_specific_metabolite('pi_e',1000);
my_layout.set_specific_metabolite('h2o_e',1000);
my_layout.set_specific_metabolite('h_e',1000);
```

8. Visualize the media composition and relevant COMETS parameters (diffusion constants, “static” and “refresh” values), which is stored as a pandas dataframe:

```
My_layout.media # this shows a pandas data.frame
```

Run the COMETS simulation

9. Define the comets object by passing the previously created layout and a parameters.

```
my_simulation = c.comets(my_layout, my_params)
```

10. Run the simulation. (Note that in this example, there will be no growth as we did not define a media allowing for it, e.g. the carbon source):

```
my_simulation.run()
```

11. Access the output of the COMETS simulation run

```
print(my_simulation.run_output) # this shows initialization and biomasses at each time step. This also shows a Java stack trace if COMETS had an internal error
```

12. Access the errors of the COMETS simulation run

```
print(my_simulation.run_errors) # should be empty if everything worked
```

The results of the successful simulation are stored in several fields in the comets object, depending on whether the parameters writeTotalBiomasslog, writeBiomassLog, writeFluxLog and writeMediaLog were set to true.

- The field total_biomass stores the total biomass (summed up over all coordinates) for each timepoint and species.
- The field biomass stores detailed biomass values for each timepoint, coordinate and species.
- The field media stores the composition of the media at each timepoint.
- The field fluxes stores the metabolic fluxes for each species, coordinate and timepoint.

Additionally, specific comets models will have additional output fields; for instance, specificMedia will contain the concentration of specific media components if set up. Similarly, if we run a simulation with evolution, the field genotypes will store information about each species such as its ancestor and which mutation it suffered.

All of the output files are stored as pandas dataframes which can be further analyzed or plotted using standard Python tools.

The basics of COMETS using the command line

COMETS is a Java application and is invoked by the following commands, typically packaged in a script file comets_scr. In Windows this file will typically have a .bat extension, comets_scr.bat.
In Windows, this file will contain the line:

```
java -classpath
"%COMETS_HOME%/lib/colt/lib/colt.jar";"%COMETS_HOME%/lib/colt/lib/concurrent.jar";"%COMETS_H
OME%/lib/jdistlib-0.4.5-bin.jar";"%COMETS_HOME%/lib/commons-lang3-3.9/commons-lang3-3.9-
sources.jar";"%COMETS_HOME%/lib/commons-lang3-3.9/commons-lang3-
3.9.jar";"%COMETS_HOME%/lib/commons-rng-1.0/commons-rng-simple-
1.0.jar";"%COMETS_HOME%/lib/commons-rng-1.0/commons-rng-sampling-
1.0.jar";"%COMETS_HOME%/lib/commons-rng-1.0/commons-rng-jmh-
1.0.jar";"%COMETS_HOME%/lib/commons-rng-1.0/commons-rng-core-
1.0.jar";"%COMETS_HOME%/lib/commons-rng-1.0/commons-rng-client-api-
1.0.jar";"%COMETS_HOME%/lib/commons-math3-3.6.1/commons-math3-
3.6.1.jar";"%COMETS_HOME%/lib/commons-math3-3.6.1/commons-math3-3.6.1-
tools.jar";"%COMETS_HOME%/lib/junit/junit-4.12.jar";"%COMETS_HOME%//lib/junit/hamcrest-core-
1.3.jar";"%GUROBI_HOME%/lib/gurobi.jar";"%COMETS_HOME%/lib/jogl/jogamp-all-platforms/jar/jogl-
all.jar";"%COMETS_HOME%/lib/jogl/jogamp-all-platforms/jar/gluegen-
rt.jar";"%COMETS_HOME%/lib/jogl/jogamp-all-
platforms/jar/gluegen.jar";"%COMETS_HOME%/lib/jogl/jogamp-all-platforms/jar/gluegen-rt-natives-linux-
amd64.jar";"%COMETS_HOME%/lib/jogl/jogamp-all-platforms/jar/jogl-all-natives-linux-
amd64.jar";"%COMETS_HOME%/lib/JMatIO/lib/jmatio.jar";"%COMETS_HOME%/lib/JMatIO/JMatIO-
041212/lib/jmatio.jar";"%COMETS_HOME%/bin/comets_2.10.0.jar" -
Djava.library.path="%GUROBI_HOME%/lib";"%GUROBI_HOME%/bin";"%COMETS_HOME%/lib/jogl/jog
amp-all-platforms/lib"  edu.bu.segrelab.comets.Comets -loader
edu.bu.segrelab.comets.fba.FBACometsLoader -script %1
```

In Linux and MacOS:

```
#!/bin/bash
if [ -z "$1" ]; then
  echo usage: $0 \"comets script name\"
  exit
fi
SCRIPT=$1

java -classpath
$COMETS_HOME/lib/colt/lib/colt.jar:$COMETS_HOME/lib/colt/lib/concurrent.jar:$COMETS_HOME/lib/jdi
stlib-0.4.5-bin.jar:$COMETS_HOME/lib/commons-lang3-3.9/commons-lang3-3.9-
sources.jar:$COMETS_HOME/lib/commons-lang3-3.9/commons-lang3-
3.9.jar:$COMETS_HOME/lib/commons-rng-1.0/commons-rng-simple-
1.0.jar:$COMETS_HOME/lib/commons-rng-1.0/commons-rng-sampling-
1.0.jar:$COMETS_HOME/lib/commons-rng-1.0/commons-rng-jmh-
1.0.jar:$COMETS_HOME/lib/commons-rng-1.0/commons-rng-core-
1.0.jar:$COMETS_HOME/lib/commons-rng-1.0/commons-rng-client-api-
```

```
1.0.jar:$COMETS_HOME/lib/commons-math3-3.6.1/commons-math3-
3.6.1.jar:$COMETS_HOME/lib/commons-math3-3.6.1/commons-math3-3.6.1-
tools.jar:$COMETS_HOME/lib/junit/junit-4.12.jar:$COMETS_HOME//lib/junit/hamcrest-core-
1.3.jar:/usr/local/share/java/glpk-
java.jar:$GUROBI_COMETS_HOME/lib/gurobi.jar:$COMETS_HOME/lib/jogl/jogamp-all-platforms/jar/jogl-
all.jar:$COMETS_HOME/lib/jogl/jogamp-all-platforms/jar/gluegen-rt.jar:$COMETS_HOME/lib/jogl/jogamp-
all-platforms/jar/gluegen.jar:$COMETS_HOME/lib/jogl/jogamp-all-platforms/jar/gluegen-rt-natives-linux-
amd64.jar:$COMETS_HOME/lib/jogl/jogamp-all-platforms/jar/jogl-all-natives-linux-
amd64.jar:$COMETS_HOME/lib/JMatIO/lib/jmatio.jar:$COMETS_HOME/lib/JMatIO/JMatIO-
041212/lib/jmatio.jar:$COMETS_HOME/bin/comets_2.10.0.jar -
Djava.library.path=$GUROBI_COMETS_HOME/lib/:$COMETS_HOME/lib/jogl/jogamp-all-platforms/lib
edu.bu.segrelab.comets.Comets -loader edu.bu.segrelab.comets.fba.FBACometsLoader  -script
$SCRIPT
```

In Linux and MacOS certainly this file may be written in any alternative shells. Here we give an example only for bash.

Comets is run on a command line by executing the comets_scr script (comets_scr.bat in Windows) which take as an argument the name of an input file, in this case named comets_script:

In Linux and MacOS by executing on the command line prompt:

```
comets_scr comets_script
```

In Windows:

```
comets_scr.bat comets_script
```

Here we assumed that the path to the directory $COMETS_HOME where Comets is installed, and where comets_scr is has been set properly, and that it has been added to the user's path. Alternatively, comets_scr can be copied in the working directory, and the above command can be executed as described above or as:

```
./comets_scr comets_script
```

The name of the comets_script file is arbitrary and may be customized. If however the above java command in the comets_scr file is typed directly on the command line, the variable $SCRIPT (%1 in Windows) should be replaced with the name of the comets_script file. Also, the variables $COMETS_HOME and $GUROBI_COMETS_HOME (%COMETS_HOME% and %GUROBI_COMETS_HOME%) must be set, to the directories where Comets and Gurobi are installed. These variables will be set during the installation process, however the most common reason for failure to launch Comets is the failure to properly set them.
The input file comets_script contains the information on the names of the two parameters and one layout input file:

```
load_comets_parameters global_params.txt
load_package_parameters package_params.txt
```

```
load_layout  layout.txt
```

Here global_params.txt, package_params.txt and layout.txt are the names of the input files that contain the parameters and the spatial layout correspondingly. The names of these files are arbitrary and may be customized. We have provided examples of these files in the text of the procedures.

The procedures for executing a simulation job therefore (as illustrated in Fig. 2 in the main text) is as follows:

1. Prepare the input files global_params.txt, package_params.txt and layout.txt in the working directory.
2. Prepare the files comets_scripts with the above names in the working directory.
3. Run comets_scr comets_script on the command line.

A warning is due here, that if the terminal where this command was executed is closed before the Comets run has finished, the job will be abruptly stopped, without finishing. One way to avoid this is to run it in the background by the nohup command in Linux and MacOS:

```
nohup comets_scr comets_script &
```

A common environment where Comets is one with a job queuing system. This is typically done on computational clusters. Here we present an example of a typical way Comets job can be submitted to a queue with the qsub command:

```
qsub -pe omp 16 -l h_rt=48:00:00 qscript
```

The two options are specifying the allocated resources for the job, in this example we are requesting 16 slots for Shared Memory applications, and a total run time of at least 48 hours. The file qscript contains the steps necessary to run Comets:

```
#!/bin/bash -l
module load gurobi/9.0.0
./comets_scr comets_script
```

The first line simply identifies the file as a bash script, the second line loads the necessary cluster module, and the third line is the actual command to run Comets, as we discussed above.

Supplementary Discussion 3: Spatial settings in the cometspy toolbox

The cometspy toolbox is capable of operating all spatial settings available in COMETS. Here, we describe the minimal requirements to run a spatial simulation, and then describe some optional methods for more complex simulations. We recommend also to spend some time looking through the api documents

available at https://cometspy.readthedocs.io/en/latest/index.html for a full list of cometspy objects and methods. Also, all methods have descriptions of usage when examined with help().

Minimal settings

For clarity, first we initialize cometspy and generate a test model using cobrapy's cobra.test module.

```
import cometspy as c
import cobra.test
ecoli_core = cobra.test.create_test_model('textbook')
ecoli = c.model(ecoli_core)
ecoli.open_exchanges()
```

1. The primary setting which must be changed to have a spatial simulation is to change the grid attribute of the layout object. This attribute contains two numbers which specify the length and width of the spatial simulation, in lattice boxes.

```
ly = c.layout()
ly.grid = (10, 10) # sets up a spatial simulation of length = 10, width = 10
```

2. Additionally, model biomass can then be specified within this spatial range. This is done by providing a list of lists to the model's initial_pop attribute. In each list, we first state the x location, then the y location, then the initial number of grams of biomass. For example, here we put 1.e-6 grams of biomass at location (x = 3, y = 5) and 2.e-7 grams of biomass at location (x = 8, y = 5):

```
ecoli.initial_pop = [[3, 5, 1.e-6], [8, 5, 2.e-7]]
```

3. Now we add that model to the spatial layout.

```
ly.add_model(ecoli)
```

4. Set the spaceWidth parameter. Just like in a single-box simulation, the spaceWidth parameter (units of cm) is what sets the volume of a box (spaceWidth ^ 3), which influences the concentration of metabolites and therefore interacts with Michaelis-Menten uptake rates. However, it is even more critical in spatial simulations, because it will change how quickly metabolites and biomass will diffuse to adjacent boxes for a given diffusion constant.

```
P = c.params()
p.set_param("spaceWidth", 0.1) # in cm
```

That is the minimal number of changes to create a spatial simulation. Metabolite concentrations will still need to be initialized. If a user sets the initial metabolite abundances using the standard set_specific_metabolite([metabolite name], [mmol]) method of the layout object, those metabolites will be put homogeneously into every box. The layout object and params object will still need to be added to a comets object to run, as usual.

Optional settings

1. Layout methods to set initial mmol amounts of metabolites in specific locations, as well as their refresh rates and constant (static) values:

layout.set_specific_metabolite_at_location(metabolite_name, location, amount)

layout.set_specific_refresh_at_location(metabolite_name, location, amount)

layout.set_specific_static_at_location(metabolite_name, location, amount)

2. Layout method to add impenetrable barriers that block diffusion and biomass motion:

layout.add_barriers(barrier_locations)

3. Layout methods that set up "regions" in which metabolite diffusion parameters and friction can differ.

layout.set_region_map(region_map)

layout.set_region_parameters(region, diffusion, friction)

4. Model method and associated params setting to use non-linear biomass diffusion:

model.add_nonlinear_diffusion_parameters()

params.set_param("biomassMotionStyle", "ConvNonlin Diffusion 2D")

5. Model method and associated params setting to use convective biomass motion:

model.add_convection_parameters()

params.set_param("biomassMotionStyle", "Convection 2D")

6. Params settings to set universal biomass diffusion constant

params.set_param("flowDiffRate", number)

7. Layout method to set all metabolites', or metabolite-specific, diffusion constants

layout.set_metabolite_diffusion(diffusion_constant)

layout.set_specific_metabolite_diffusion(metabolite_name, diffusion_constant)

8. Params settings to save spatially-explicit data:

```
params.set_param("writeBiomassLog", True)
params.set_param("BiomassLogRate", number)
params.set_param("writeMediaLog", True)
params.set_param("MediaLogRate", number)
params.set_param("writeFluxLog", True)
params.set_param("FluxLogRate", number)
```

9. Access to saved results

```
comets.biomass # is a pandas dataframe
comets.media # is a pandas dataframe
comets.fluxes_by_species # is a dictionary with model ids as keys and pandas dataframes as values
```

Supplementary Discussion 4: Detailed structure of the Output Files

Console output

The standard console output of COMETS is either displayed on the GUI console or saved in an output file if run on a queueing system. The console output format is typically:

```
-script
running script file: comets_script
Loading layout file 'layout.txt'...
Found 2 model files!
Loading 'e_coli_core1.txt' ...
Loading 'e_coli_core1.txt' ...
Done!
 Testing default parameters...
Done!
Optimizer status code = 5 (looks ok!)
objective solution = [D@99e8be2
Loading 'e_coli_core2.txt' ...
Loading 'e_coli_core2.txt' ...
Done!
 Testing default parameters...
Done!
Optimizer status code = 5 (looks ok!)
objective solution = [D@7f1a75d
Constructing world...
Done!
medialist ac[e] acald[e] akg[e] co2[e] etoh[e] for[e] fru[e] fum[e] glc__D[e] gln__L[e] glu__L[e] h2o[e] h[e] lac__D[e] mal__L[e] nh4[e] o2[e] pi[e] pyr[e] succ[e]
WRITING MEDIA LOG
Cycle 1
Total biomass:
Model 0: 2.5117253201512343E-6
Model 1: 2.51172552547532E-6
Cycle complete in 0.695s
Cycle 2
Total biomass:
Model 0: 2.52350543670516E-6
Model 1: 2.523505417783118E-6
Cycle complete in 0.254s
…
Cycle 10000
Total biomass:
Model 0: 1.8129744692017875E-5
Model 1: 1.8466722853528566E-5
WRITING MEDIA LOG
Cycle complete in 0.315s
Cycle 10001
End of simulation
Total time = 1343.506s
```

In addition to the console output, if errors are detected, they are written in the standard error output file. The possible error messages are documented in the Troubleshooting section.

Output files

The generation of the output file is optional and is controlled with the corresponding parameter in the global parameters input file. The rate of the output recording is also controlled by an input parameter.

Total biomass file.

This is a space-delimited text format file with the first column containing the simulation step, and the integrated total biomass for each model in separate columns:

```
0    5E-6
1    5.0234319275E-6
2    5.0469736628E-6
3    5.0706257205E-6
4    5.0943886175E-6
```

The biomass unit is grams.

Biomass file.

This is a MATLAB .m format file with the record of the spatial layout of the biomass. The variable is of the following format:

```
biomass_<step>_<model> (<xcoordinate>,<ycoordinate>)= <amount>;
```

Example of a biomass file of a 100x100 points layout, containing two models with initial population at the center of the layout. The biomass recording rate is each 100 simulation steps.

```
biomass_0_0 = sparse(100, 100);
biomass_0_0(51, 51) = 2.5E-6;
biomass_0_1 = sparse(100, 100);
biomass_0_1(51, 51) = 2.5E-6;
biomass_100_0 = sparse(100, 100);
biomass_100_0(51, 51) = 3.5426406048E-6;
biomass_100_1 = sparse(100, 100);
biomass_100_1(51, 51) = 3.4688274362E-6;
```

The biomass unit is grams.

Media file

The media file is a MATLAB .m file format with the record of all external metabolite amounts in mmol units. The first line is an array of all metabolite names. The format is:

media_<time>{<metabolite index>}(<xcoordinate>, <ycoordinate>) = <amount>;

media_names = { 'ac[e]', 'acald[e]', 'akg[e]', 'co2[e]', 'etoh[e]', 'for[e]', 'fru[e]', 'fum[e]', 'glc__D[e]', 'gln__L[e]', 'glu__L[e]', 'h2o[e]', 'h[e]', 'lac__D[e]', 'mal__L[e]', 'nh4[e]', 'o2[e]', 'pi[e]', 'pyr[e]', 'succ[e]'};
media_0{1} = sparse(zeros(100, 100));
media_0{2} = sparse(zeros(100, 100));
media_0{3} = sparse(zeros(100, 100));
…
media_10000{18}(100, 97) = 1E0;
media_10000{18}(100, 98) = 1E0;
media_10000{18}(100, 99) = 1E0;
media_10000{18}(100, 100) = 1E0;
media_10000{19} = sparse(zeros(100, 100));
media_10000{20} = sparse(zeros(100, 100));

Fluxes file

This MATLAB .m format file contains the record of all fluxes of all models in each spatial point for a recorded time. The format is:

fluxes{<time>}{<x coordinate>}{<y coordinate>}{<model index>} = [<flux values>];

fluxes{10}{1}{1}{1} = [-1.5084028022E1 0E0 -1.4742953314E1 5.0561349588E-1 5.0561349588E-1 -1.4742953314E1 0E0 0E0 0E0 -1.5084028022E1 8.39E0 -1.0840773838E1 4.6863796078E-1 0E0 5.0561349588E-1 0E0 0E0 3.5002712702E1 -1.5084028022E1 1.4742953314E1 -0E0 -0E0 0E0 1.5084028022E1 3.1251640737E1 -0E0 -0E0 -1.8474787691E1 -0E0 -0E0 5.5395471544E1 -1.167028166E1 -0E0 -0E0 -2.5553890725E0 -0E0 -1.7239784663E0 -0E0 -0E0];

Complete record file

This is a file in the MATLAB .mat format that contains all the input and output information for a given COMETS run. This file can be of a very large size and is meant to be used only for archiving purposes.

Matfile_example =

struct with fields:

allowCellOverlap: 'true'
deathRate: 0
defaultHill: 1
defaultKm: 0.0100
defaultVmax: 10
exchangestyle: 'Monod Style'
flowDiffRate: 3.0000e-10
flux: [5-D double]
growthDiffRate: 0
maxSpaceBiomass: 8.8000e-06
minSpaceBiomass: 1.0000e-10
numDiffPerStep: 10
numRunThreads: 10
showCycleCount: 'true'
showCycleTime: 'true'
spaceWidth: 0.0200
timeStep: 0.0100
timeStepsSaved: [11×1 double]
toroidalWorld: 'false'
total_biomass: [11×2 double]

Supplementary Discussion 5: Setting the environmental variables in Mac OS X

The environment variables for COMETS and Gurobi can be manually set on Mac OS X using plist files, which are loaded at login by the system's launchctl service. Three plist files are required to run COMETS using the Matlab toolbox: one for COMETS_HOME, and two for Gurobi (one for GUROBI_HOME and one for the license file). These can be created and saved as follows:

1. Quit MATLAB

2. Open a text editor and paste the following commands into a blank file:

```
<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
   <plist version="1.0">
   <dict>
   <key>Label</key>
   <string>ENVIRONMENT_VARIABLE</string>
   <key>ProgramArguments</key>
   <array>
     <string>/bin/launchctl</string>
     <string>setenv</string>
```

```
    <string>ENVIRONMENT_VARIABLE</string>
    <string>ENVIRONMENT_VARIABLE_LOCATION</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
</dict>
</plist>
```

3. For each of the three files, replace '**ENVIRONMENT_VARIABLE**' and '**ENVIRONMENT_VARIABLE_LOCATION**' as follows:

ENVIRONMENT_VARIABLE	**ENVIRONMENT_VARIABLE_LOCATION**
COMETS_HOME	/Applications/COMETS
GUROBI_HOME	/Library/gurobi902/mac64
GRB_LICENSE_FILE	/ Library/gurobi902/gurobi.lic

Note: *the directories listed above are the default installation locations for COMETS and Gurobi. These locations must be changed in the plist files being generated if COMETS or Gurobi were installed in different directories.*

4. Save each of the plist files, e.g. as 'cometspath.plist,' 'gurobipath.plist,' and 'gurobilicensepath.plist' in the directory ~/Library/LaunchAgents/.

5. Either reboot your Mac to load the plist files, or load them manually as follows:

    a. Open Terminal and run the following commands:

```
launchctl load ~/Library/LaunchAgents/cometspath.plist
launchctl load ~/Library/LaunchAgents/gurobipath.plist
launchctl load ~/Library/LaunchAgents/gurobilicensepath.plist
```

    b. Quit then reopen Terminal. You can run the command `printenv` to verify that the environment variables exist.

6. Open MATLAB and use the `getenv` function (e.g. `getenv('COMETS_HOME')`) to verify that the environment variables have been loaded.

Supplementary Table 1: Table of COMETS input parameters, with units, default value and a short definition.

Simulation parameters

Parameter	**Unit**	**Default value**	**Definition or notes**
timeStep	hour	1.0	The amount of time between two consecutive simulation updates.
spaceWidth	cm	0.1	Width of one side of the 3d box in the 2D or 3D grid. Therefore, volume of a box = spaceWidth^3. Warning: this value matters for molarity calculations.
maxCycles	steps	Unlimited	Number of DFBA iterations (steps) for the simulation. The total simulation time will be timeStep * maxCycles.
deathRate	fraction/timepoint	0.1	The rate of biomass removal per time step.
maxSpaceBiomass	gr	10	Maximum biomass allowed in one grid box.
minSpaceBiomass	gr	1e-10	Minimum biomass in one grid box not considered zero.
cellSize	gr	4.3e-13	Grams in one cell. Relevant in simulations with serial dilutions or mutations.

exchangeStyle	Standard FBA, Monod Style, Pseudo-Monod Style	Standard FBA	The uptake function for the exchange reactions.
defaultVmax	mmol $(gCDW)^{-1}$ $(hour)^{-1}$	10	Default maximum uptake rate of a metabolite for the Monod Style exchange. This overrides exchange reaction boundaries with greater magnitude, when using Monod updating.
defaultKm	mmol $(cm^3)^{-1}$	5	Default concentration of a metabolite in which uptake is half-maximal. This value is compared with the metabolite concentration / $spaceWidth^3$ when computing Monod uptake.
defaultHill		2	Hill coefficient. Alters the shape of the Monod uptake curve.
defaultAlpha	1/(mmol $(cm^3)^{-1}$)	1	The default Alpha coefficient (slope) for the Pseudo-Monod style exchange.
defaultW	mmol $(gCDW)^{-1}$ $(hour)^{-1}$	10	The default W coefficient (plateau) for the Pseudo-Monod style exchange.
minConcentration	mmol $(cm^3)^{-1}$	1e-26	Minimal

			concentration of metabolites in the media.
numRunThreads		1	If >1, allow multithreaded computation. The number of threads to run in parallel.
numDiffPerStep		10	Number of substeps of media diffusion per biomass update step.
allowCellOverlap		FALSE	If true, allows different species to occupy the same space.

Parameters related to spatial propagation of either biomass or metabolites

Parameter	**Units/Values allowed**	**Default**	**Description**
biomassMotionStyle	Diffusion 2D(Crank-Nicolson), Diffusion 2D(Eight Point), Diffusion 3D, Convection 2D, Convection 3D, ConvNonlin Diffusion 2D	Diffusion 2D(Crank-Nicolson)	Sets the method used for propagation of biomass. Only one of the indicated strings is an allowed value.
growthDiffRate	cm2/s	1.00E-07	The default diffusion constant for the actively growing biomass in the Diffusion 2D (CN and EP) model.
flowDiffRate	cm2/s	1.00E-07	The default diffusion constant for the non-growing biomass in the Diffusion 2D (CN

			and FP) model.
defaultDiffConst	cm2/s	1.00E-05	The default diffusion constant for extracellular metabolites.

Parameters related to log file writing

Parameter	**Default**	**Description**
useLogNameTimeStamp	TRUE	If TRUE, appends a time stamp to every log file name.
writeFluxLog	FALSE	If true, writes fluxes out to a log file.
fluxLogName	flux_log.txt	The name of the flux log file.
fluxLogRate	1	How often to write to the flux file (number of simulation steps). A value of 1 will cause writing after every step.
writeMediaLog	FALSE	If true, writes media information to a log file.
mediaLogName	media_log.txt	The name of the media log file.
mediaLogRate	1	How often to write to the media file.
writeSpecificMediaLog	FALSE	If true, writes the media log only for the metabolites specified by specificMedia parameter.
specificMediaLogName	specific_media_log.txt	The name of the specific media log file.
specificMedia		Names of metabolites for which we want to store media.
writeBiomassLog	FALSE	If true, writes biomass information to a log file.
biomassLogName	biomass_log.txt	The name of the biomass log file.
biomassLogRate	1	How often to write to the biomass file.
writeTotalBiomassLog	FALSE	If true, writes a summation of all biomass information to a log file.

totalBiomassLogName	total_biomass_log.txt	The name of the total biomass log file.
totalBiomassLogRate	1	How often to write to the total biomass log file.

Parameters related to graphical user interface and image caption

Parameter	**Default**	**Description**
showGraphics	TRUE	If true, the image will be displayed.
colorRelative	TRUE	If true, colors each space relative to the space with the highest value.
showCycleTime	TRUE	If true, shows the time it took to finish the fba cycle in the output.
showCycleCount	TRUE	If true, shows the current cycle number in the output.
pauseOnStep	TRUE (false if running a script)	If true, pauses the simulation after completing a step.
displayLayer	0	Sets the current medium component (or biomass) to be displayed. The user must determine the number of the medium or biomass from the layout.
pixelScale	4	The number of pixels to render for each space.
saveSlideshow	FALSE	If true, saves a graphics slideshow to a series of files.
slideshowName	“/path_to_directory/slideshow”	The header of the names and path of the files with saved images. The format is "name"_number.slideshowExt

slideshowColorValue	10	Sets the color of the biomass when creating and saving an image.
colorRelative	TRUE	Show the colors relative to each model, i.e. on an RGB palette.
slideshowColorRelative	TRUE	As colorRelative above, applied to the slideshow.
slideshowRate	1	The number of steps between taking a slideshow picture.
slideshowLayer	0	Sets the current medium component (or biomass) to be displayed. The user must determine the number of the medium or biomass from the layout.
slideshowExt	png	The file extension(format) for slideshow pictures. Currently, “png” “bmp” and “jpg” are supported. “png” is recommended.
barrierColor	0xff7D7D7D (gray)	Barrier color in hex.
backgroundColor	0xff000000 (black)	Background color in hex.

Parameter related to the extracellular reactions model

numExRxnSubsteps		12	Number of extracellular reactions substeps per biomass update step.

Parameters related to lag phases

Parameter	**Unit**	**Default value**	**Definition or notes**
simulateActivation		FALSE	If true, the models are activated with

			the set activation rate.
activateRate	h^{-1}	0.001	The value of activation rate.

Parameters related to specific modes of growth, such as serial dilutions or chemostat mode

Parameter	**Unit**	**Default value**	**Definition or notes**
batchDilution		FALSE	Whether to perform serial dilutions.
dilFactor	Dil. factor	1e-2	If >1, dilution factor; if <1, 1/dilution factor.
dilTime	h	12	Periodicity of serial dilutions.
metaboliteDilutionRate	Fraction per hour	0	The rate of dilution of a metabolite.

Parameters related to evolution (mutations)

Parameter	**Unit**	**Default value**	**Definition or notes**
evolution		FALSE	If true, the simulation will perform mutations.
mutRate	Per genome and cycle	1e-9	Mutation rate for reaction deletions.
addRate	Per genome and cycle	1e-9	Mutation rate for reaction additions.

Parameters related to genome size cost

Parameter	**Unit**	**Default value**	**Definition or notes**
costlyGenome		FALSE	Does genome size penalize growth.
geneFractionalCost		0	How much does

			genome size penalize growth. The cost grows exponentially with genome size, with an exponent of 2.

Additional, less often used general simulation parameters

Parameter	**Units**	**Default**	**Description**
toroidalWorld		FALSE	If true, creates periodic boundary conditions.
showCycleCount		TRUE	If true, shows the current number of cycles/steps on the console.
showCycleTime		TRUE	If true, shows the time of a cycle/step on the console.
randomSeed		0	Seed value for the semi-random number generator.
defaultVelocityVector	cm/s	(0,0,0)	The default value for the velocity vector in the flow model. .
writeVelocityLog		FALSE	If true, writes velocity information to a log file.
velocityLogRate		1	How often to write to the velocity file (number of simulation steps). A value of 1 will cause writing after every step.
velocityLogName		velocity_log.txt	The name of the

			velocity log file.
writeMatFile		FALSE	If true, writes all of the simulation information to a log file.
matFileName		comets_log.mat	The name of the .mat log file.
matFileRate		1	How often to write to the .mat file (number of simulation steps). A value of 1 will cause writing after every step. Warning: writing to this file every step may result in very large .mat file.
biomassLogFormat		MATLAB	The format in which the log file will be written. The default is .m MATLAB file. If the value is COMETS, the output is written in a space separated file.
mediaLogFormat		MATLAB	The format in which the log file will be written. The default is .m MATLAB file. If the value is COMETS, the output is written in a space separated file.
fluxLogFormat		MATLAB	The format in which the log file will be written. The default is .m MATLAB file. If the value is COMETS, the output

			is written in a space separated file.
velocityLogFormat		MATLAB	The format in which the log file will be written. The default is .m MATLAB file. If the value is COMETS, the output is written in a space separated file.

Model-specific, these parameters are specified in the model file

Parameter	**Unit**	**Default value**	**Definition or notes**
Optimizer	GUROBI, GLPK	GUROBI	The optimizer to be used for solving the FBA optimization.
OBJECTIVE_STYLE	MAX_OBJECTIVE_MIN_TOTAL, MAX_OBJECTIVE	MAX_OBJECTIVE	The type of optimization to be used.
VMAX_VALUES	mmol $(gCDW)^{-1}$ $(hour)^{-1}$	Same as the global default.	Maximum flux constant for the Michaelis-Menten type exchange, for each reaction. Each reaction can be assigned separate value in the model file.
KM_VALUES	$mmol/cm^3$	Same as the global default.	The Michaelis constant for the Michaelis-Menten type exchange, for each reaction. Each reaction can be assigned separate value in the model file.

packedDensity	g/cm^3	1.0	The biomass density of densely packed cells in the CONVECTION 2D model.
frictionConst	Pa sec/cm^2	1.0	The friction constant in the Convection 2D model.
elasticModulus	Pa	1.0	The elastic constant in the Convection 2D model.
convDiffConstant	cm^2/s	1.0	The diffusivity constant in the Convection 2D model.
convNonlinDiffZero	cm^2/s	1.0	The linear diffusivity in the ConvNonlin Diffusion 2D model.
convNonlinDiffN	N/A It is a function of convNonlinDiffExponent	1.0	The non-linear diffusivity coefficient in the ConvNonlin Diffusion 2D model.
convNonlinDiffExponent		1	The exponent in the ConvNonlin Diffusion 2D model.
convNonlinDiffHillN		10	The exponent in the Hill function model of local growth dependent diffusivity.
convNonlinDiffHillK	N/A It is a function of convNonlinDiffHillN.	0.9	The K constant in the Hill function model of local growth dependent diffusivity.
noiseVariance		0.0	The variance of the growth noise factor.

neutralDrift		FALSE	The boolean switch for the demographic noise.
neutralDriftSigma	$g^{1/2}\ s^{-1}$		The prefactor constant for the demographic noise.

1. Orth, J. D., Thiele, I. & Palsson, B. Ø. What is flux balance analysis? *Nat. Biotechnol.* **28**, 245–248 (2010).
2. O'Brien, E. J., Monk, J. M. & Palsson, B. O. Using Genome-scale Models to Predict Biological Capabilities. *Cell* **161**, 971–987 (2015).
3. Heirendt, L. *et al.* Creation and analysis of biochemical constraint-based models using the COBRA Toolbox v.3.0. *Nat. Protoc.* **14**, 639–702 (2019).
4. Feist, A. M. & Palsson, B. O. The biomass objective function. *Current Opinion in Microbiology* vol. 13 344–349 (2010).
5. Ibarra, R. U., Edwards, J. S. & Palsson, B. O. Escherichia coli K-12 undergoes adaptive evolution to achieve in silico predicted optimal growth. *Nature* **420**, 186–189 (2002).
6. Fong, S. S. & Palsson, B. Ø. Metabolic gene–deletion strains of Escherichia coli evolve to computationally predicted growth phenotypes. *Nat. Genet.* **36**, 1056–1058 (2004).
7. Segrè, D., Vitkup, D. & Church, G. M. Analysis of optimality in natural and perturbed metabolic networks. *Proc. Natl. Acad. Sci. U. S. A.* **99**, 15112–15117 (2002).
8. Wintermute, E. H., Lieberman, T. D. & Silver, P. A. An objective function exploiting suboptimal solutions in metabolic networks. *BMC Syst. Biol.* **7**, 98 (2013).
9. Lewis, N. E. *et al.* Omic data from evolved E. coli are consistent with computed optimal growth from genome-scale models. *Mol. Syst. Biol.* **6**, 390 (2010).
10. Fong, S. S. & Palsson, B. Ø. Metabolic gene–deletion strains of Escherichia coli evolve to

computationally predicted growth phenotypes. *Nat. Genet.* **36**, 1056–1058 (2004).

11. Mahadevan, R., Edwards, J. S. & Doyle, F. J., 3rd. Dynamic flux balance analysis of diauxic growth in Escherichia coli. *Biophys. J.* **83**, 1331–1340 (2002).
12. Harcombe, W. R. *et al.* Metabolic resource allocation in individual microbes determines ecosystem interactions and spatial dynamics. *Cell Rep.* **7**, 1104–1115 (2014).
13. Matsushita, M. *et al.* Interface growth and pattern formation in bacterial colonies. *Physica A: Statistical Mechanics and its Applications* vol. 249 517–524 (1998).
14. Farrell, F. D. C., Hallatschek, O., Marenduzzo, D. & Waclaw, B. Mechanically driven growth of quasi-two-dimensional microbial colonies. *Phys. Rev. Lett.* **111**, 168101 (2013).
15. Tronnolone, H. *et al.* Diffusion-Limited Growth of Microbial Colonies. *Scientific Reports* vol. 8 (2018).
16. Lacasta, A. M., Cantalapiedra, I. R., Auguet, C. E., Peñaranda, A. & Ramírez-Piscina, L. Modeling of spatiotemporal patterns in bacterial colonies. *Physical Review E* vol. 59 7036–7041 (1999).
17. Kozlovsky, Y., Cohen, I., Golding, I. & Ben-Jacob, E. Lubricating bacteria model for branching growth of bacterial colonies. *Phys. Rev. E Stat. Phys. Plasmas Fluids Relat. Interdiscip. Topics* **59**, 7025–7035 (1999).
18. Giverso, C., Verani, M. & Ciarletta, P. Branching instability in expanding bacterial colonies. *Journal of The Royal Society Interface* vol. 12 20141290 (2015).
19. Vassallo, L., Hansmann, D. & Braunstein, L. A. On the growth of non-motile bacteria colonies: an agent-based model for pattern formation. *The European Physical Journal B* vol. 92 (2019).
20. Ben-Jacob, E. *et al.* Generic modelling of cooperative growth patterns in bacterial colonies. *Nature* vol. 368 46–49 (1994).
21. Henrichsen, J. Bacterial surface translocation: a survey and a classification. *Bacteriological Reviews* vol. 36 478–503 (1972).

22. Müller, J. & Van Saarloos, W. Morphological instability and dynamics of fronts in bacterial growth models with nonlinear diffusion. *Phys. Rev. E Stat. Nonlin. Soft Matter Phys.* **65**, 061111 (2002).
23. Dornic, I., Chaté, H. & Muñoz, M. A. Integration of Langevin equations with multiplicative noise and the viability of field theories for absorbing phase transitions. *Phys. Rev. Lett.* **94**, 100601 (2005).
24. William H. Press. *Numerical Recipes in C: The Art of Scientific Computing*. (Cambridge University Press, 1992).
25. LeVeque, R. J. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. (SIAM, 2007).